Sub Phase:

High-Level Design

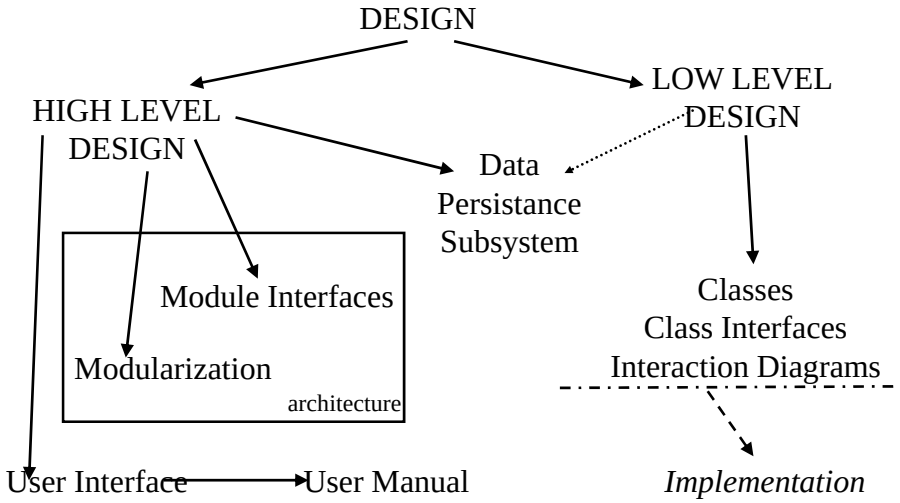
1

From Requirements Analysis to User Manual

System Design

2

Map of design phase



Activities for high level design

- Decide on an architecture model to work with
- Modularize
  - Divide your application into subsystems/modules
- Design your user interface
- If data persistence is required
  - Determine which data needs to be persistent
  - Decide on a solution for dealing with that persistent data.
- Refine your use cases
  - Add User interface interaction details
  - Add Database solution interaction details
- Describe the interfaces between subsystems

Objectives of System Design

- The design phase produces a system model
  - That is based on specific design goals for the designers
  - That defines architecture and Subsystem design
    - Identifying subsystems/modules (manageable parts)
    - Identifying architecture (hardware/software)
    - Data management / mapping
    - Access control, flow control (sequencing operations)
  - That describes boundary use cases:
    - Initialization, termination, configuration, exception handling

Design goals

- When assessing design goals consider
  - Selection of existing components (off the shelf modules or components)
  - Hardware / software mapping,
    - Are there multiple nodes or systems
    - What is each node responsible for
  - selecting solutions for managing persistent data
  - Access control policies
  - Control flow on a solution wide basis
  - Boundary conditions (startup, error, shutdown)

Design goals, System decomposition

- Identify design goals (choose aspects of the system to be optimized) Design goals are often derived from non-functional requirements.
  - Guide designers in assessing trade offs
- Develop and refine a subsystem decomposition that satisfies the maximum number of design goals and or the most critical design goals
- Refine the decomposition to better satisfy the design goals

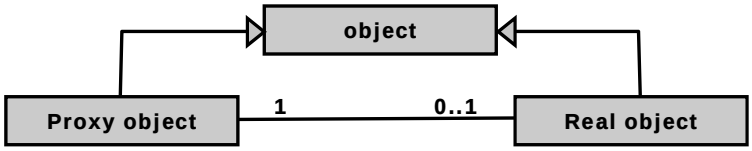
Selecting platform (mapping 1)

- First need to decide if the system resides on one computer or on multiple computers (in the same location or separated and connected through the Internet)
- Next need to select an appropriate hardware configuration, and operating system (virtual machine)
- Next need to select any other external software components (database package, communication package ...)

Allocating subsystems (mapping 2)

- Next we consider the subsystems (modules) in our system and allocate modules to appropriate nodes (virtual machines)
- If objects belong to more than one module then they may need to communicate between modules or even between virtual machines
- This communication can often be supported and made more efficient using the “proxy” design pattern

Proxy Pattern



- Protection or Image proxy: enhances efficiency.
- Proxy object is similar (same interface) to real object, only contains items like checks for permission to use methods, or specifications of image
- Only when permission is allowed or the actual image is accessed is the real object used

Access Control

- In a multi user system it is common for different users (actors) to have access to different functions of the system or different parts of the data
- In most general case need to identify which methods of any shared object can be accessed by each sharing user
- Have a matrix of access information (class or method on one axis, user type on the other

Access Control: representation

- Depending on the amount and type of flexibility (generality) needed we can represent the access matrix as
  1. A global access table, explicitly list access (or no access) for each operation, class, user tuple
  2. An access control list to control if an object (method in an object) can be accessed by a particular user (e.g. Guest List)
  3. A capability to indicate if a user may access a particular method or object (e.g. Invitation)

Access Control: representation

- Depending on the amount and type of flexibility (generality) needed we can represent the access matrix as
  4. When the number of actors or objects is large methods 1, 2, 3 may not be practical. In such a case set of rules can be used to control access (e.g a firewall)
- Before using the access control the user must be authenticated to the system. Usually use username and password known only by the user themselves.
- Login over an unsecured network may also need encryption to assure security and prevent forging of login packets

Design of Control Flow

- Three approaches
  - Procedure driven
    - Procedures wait for input from user
    - Not generally compatible with OO languages
  - Event driven
    - Main loop waits for events, sends them to objects
  - Threads
    - Concurrent (multiple processes at one time) version of procedure driven

Identify boundary conditions

- Configuration: when is each object created or destroyed?
- How is each component of the system started?
- How is each component of the system shut down?
- How does the system deal with error conditions / exceptions?

Is data persistence required?

- Are there pieces of information that need to be 'remembered' when the application is not actually in use?
  - Persistent data outlive a single execution of the software system
- What data need to be persistent?
- We know some of our data should be persistent.
  - How to we assure the data is persistent?
  - Store data between executions of the software system. How?

Example: 1 data persistence

- You are designing an application which supports users in writing documents (a word processor)
  - Users do not wish to type their entire document each time they use the word processor
  - The word processor creates a file
  - The file is saved when the user leaves the word processor
  - The file is loaded into the word processor when a user wishes to continue with editing the document.
- The data (file containing document) is persistent. It is not necessary for the word processor to be running for the file to exist.

Data Persistence

- Persistent data outlive a single execution of the software system
  - Done by storing data
    - In files on external storage medium
    - In database ….

Data Persistence

- Does our application need data persistence?
  - Which data need to be persistent?
  - How should this data be stored?
    - How is this external storage medium managed?
    - What tools are to be used to manage the persistent data?
    - Can all data reside in memory at once?

LMS example: Data persistence

- What data need to be persistent?
  - Patron
  - Resource
  - Librarian
- What data do not need to be persistent?
  - Overdue fine information
  - Overdue letters
  - Search results

Flat Files

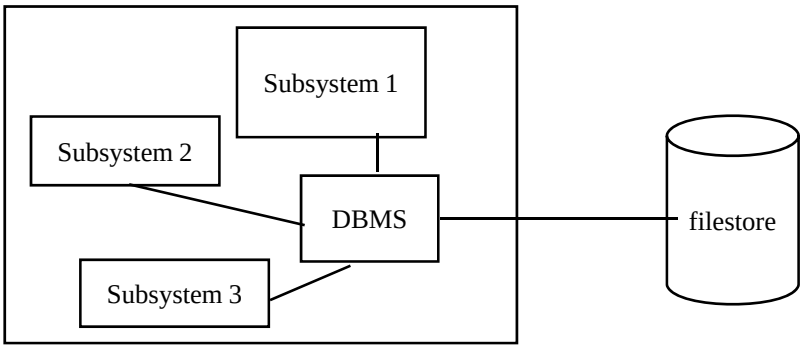
- Storage abstractions provided by OS
- Application
  - stores data as sequence of bytes
  - defines how and when data is retrieved/updated
- Low level data storage
  - Advantage: give application control over data storage (e.g., size)
  - Disadvantage: application must take care of many issues: concurrent accesses, recovery from system crash.

Database Management System DBMS

- Security (hacker proof)
- Recovery from crash
- Information growth
  - Solution still works with increased data volume
- Concurrency
  - Concurrency solution allows for increased users
- Software systems that operate on data can do so in a consistent fashion
- Efficient queries on large data set

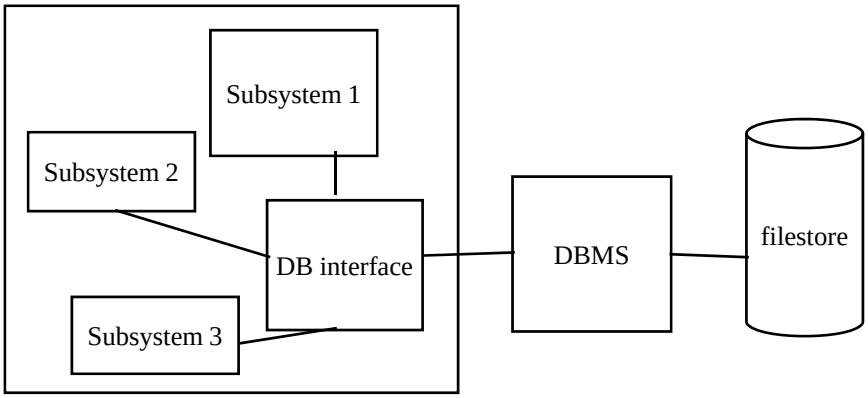
Context of a database

Self Implemented Data Storage Solution
(Part of application)



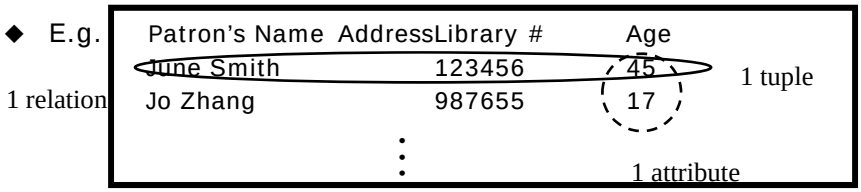
Context of a database

Commercial DBMS
(Used by application, not directly part of application)



Relational DBMS

- Data stored in structured tables (called relations)
  - Each row is a record (called a tuple), each tuple contains the values of a list of attributes
  - Each column, (called an attribute) contains the value of that particular attribute for each tuple in the relation
  - Several tuples in diff. tables may contain attributes of individual object (not object based)
  - E.g.

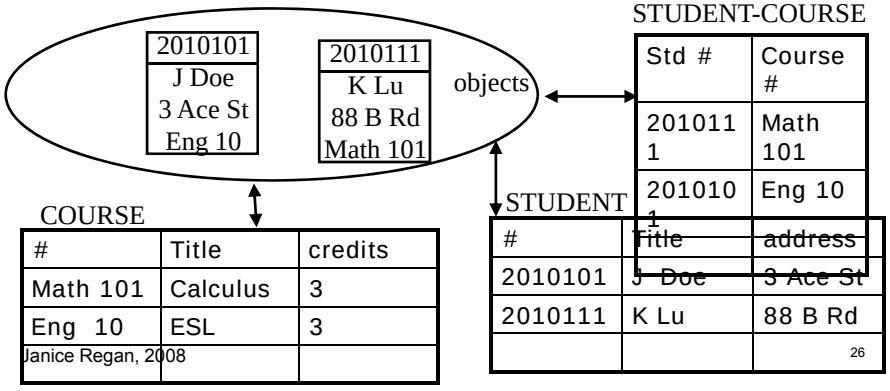


- Advantage: mature technology, stable application
- Disadvantage: high cost, performance bottleneck, not object oriented

Object Persistence

Problem:

How to go back and forth between objects held in memory and storage entities on disk when these storage entities are not object-oriented?



Object-Oriented DBMS

- Provides services similar to relational DB
- Stores data as objects and associations
- Higher level of abstraction
  - no need to translate between objects and relations (advantage)
  - provides inheritance and abstract data types
- Disadvantages: slower than relational DB, less mature technology

What solution is appropriate?

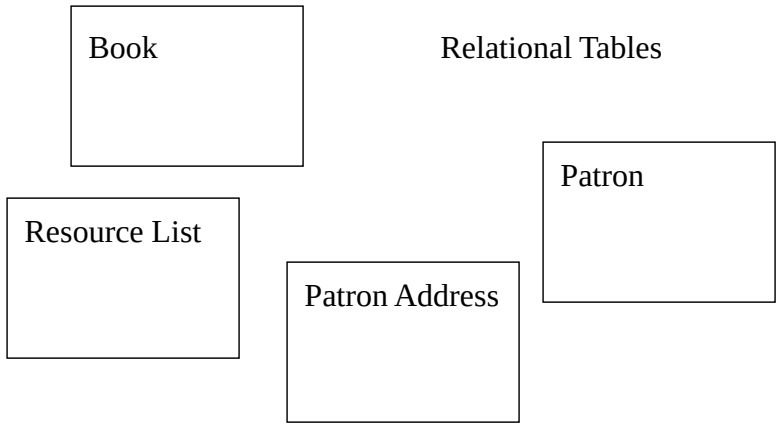
- Choose a file when
  - Voluminous data (e.g. images)
  - Low information density (e.g. archives, logs)
- Choose a database when
  - Concurrent access
  - Multiple applications over same data
- Choose a relational database when
  - Large data set
  - Complex queries over attributes
- Choose OO database when
  - Medium size data set
  - Irregular associations among objects

Based on p 269 of Bruege & Dutoit

LMS Case Study: Object Persistence

- LMS may contain 1000's of resources
- These records cannot all "live" in memory
- A DBMS is called for to provide
  - Concurrent access by multiple users
  - Security enforcement of different access levels for various user categories
  - Allow for increased data capacity

LMS Case Study:
Relational Representation of Data

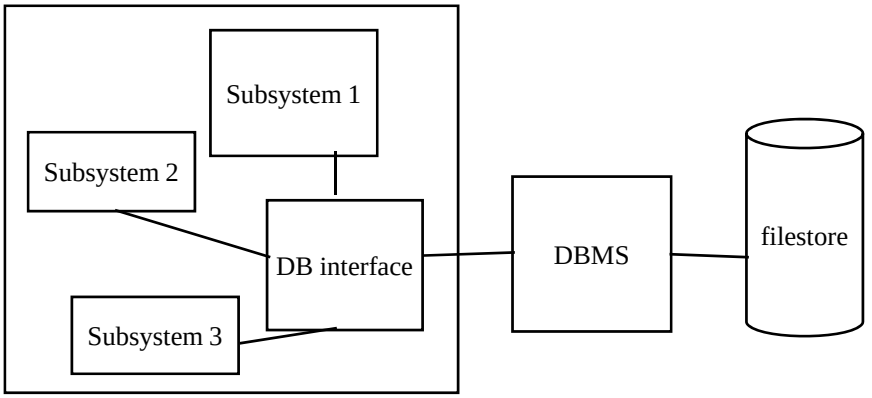


Encapsulating Data Storage

- Once storage mechanism is selected, encapsulate it into a module, then define interface between that module and the storage mechanism
  - Open Database Connectivity (ODBC) application programming interface (API)
  - Java Database Connectivity (JDBC) application programming interface (API)
  - Both use Structured Query Language (SQL) as their database access language

Context of a database

Commercial DBMS
(Used by application, not directly part of application)



Encapsulating Data Storage

- Advantage: Increase changeability
  - Allows easy substitution of different data stores
  - Makes your solution vendor independent
- Disadvantage: Reduce performance

High Level Design : Summary

- Divide your software system into subsystems
- Design your user interface
- Decide on data persistence solution
- Refine your use cases by adding
  - UI details
  - DB details
- Describe inter-subsystems interfaces

"Every module ... is characterized by its knowledge of a design decision which it hides from all others. Its interface or definition was chosen to reveal as little as possible about its inner workings."
[Parnas, Communications of the ACM 1972]

Data Persistence: Solution 1

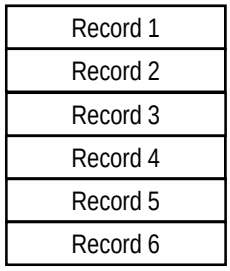
- Build DB subsystem (classes) that encapsulates disk files
  - Random access file I/O library -> file handle
  - Tutorial: Binary Random-Access I/O in Java on Course Web Site
  - File contains unsorted records of fixed length – why?
- Build public interface for that class
  - Must provide adequate functionality (get, add, remove, ...)

Data Persistence: Solution 2

- Advantage: can change file format and organization
  e.g. could use sorted records or B+ Tree
  rest of software system unaffected by this change
- How to transform objects into relational records of fixed length
  - Normalization
    - 1st normal form: fixed length records
    - 2nd normal form: remove partial dependencies
    - 3rd normal form: remove transitive dependencies

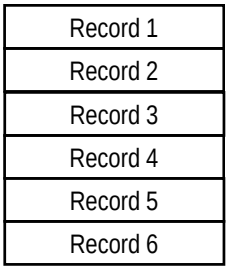
Why fixed length records?

- Minimizes fragmentation of the file.
  - If record 2 is removed, a unused space in the file will be created
  - Any new record may be put in that unused space, because all records have the same length the new record completely fills the space created by removal of the previous record
  - Fragmentation is not a problem.



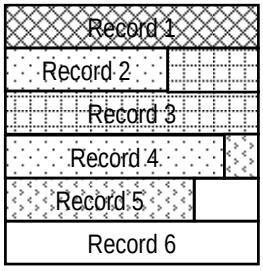
Why fixed length records?

- No need to have special support to find the beginning of the next record.
  - If you know the record length you know where each record begins. Can easily find any record (record n starts a n-1*length)



Without fixed length records

- Fragmentation becomes a problem.
  - If record 2 is removed, a unused space in the file will be created
  - The next new record may be put in that unused space if it is small enough, the new record may not fill the gap. Gaps are created between records → fragmentation



Without fixed length records

- Special support to find the beginning of the next record.
  - You must keep track of where each record begins and ends. Need a table of contents, or must step through all records